УДК 004.021

В.А. Чепурко
Державний університет інформатики і штучного інтелекту, м. Донецьк, Україна
kven@inbox.ru

Мінімізація орієнтованих детермінованих графів з ациклічними підграфами

Графи з поміченими вершинами є однією з основних моделей у розгляді проблем, пов'язаних з аналізом оперативного середовища та агентами, що рухаються по ньому, а також проблем, пов'язаних з перевіркою програм. Задача мінімізації полягає в знаходженні розбиття всіх вершин графа на класи еквівалентних. Запропоновано новий алгоритм мінімізації для графів з поміченими вершинами. Алгоритм виконує правильне розбиття на класи еквівалентних вершин.

Вступ

Графи з поміченими вершинами є однією з основних моделей при розгляді двох напрямків. Перший – це задачі, пов'язані з аналізом операційного середовища за допомогою блукаючих по ньому агентів (мобільних роботів, автоматів, пошукових програм тощо), що відносяться до категорії традиційних завдань штучного інтелекту. Однією з центральних і актуальних як у теоретичному, так і в прикладному аспектах проблем, що виникають при дослідженнях взаємодії автоматів та операційного середовища, є проблема аналізу або розпізнавання властивостей цього середовища при різній апріорній інформації і при різних способах взаємодії автомата та операційного середовища. У таких задачах розглядаються різні геометричні моделі середовищ.

Операційне середовище розглядається як граф з поміченими вершинами [1], [2]. Такі графи виникли спочатку як блок-схеми та схеми програм, а в даний час знаходять застосування в задачах навігації роботів [3].

Другий напрямок – це блок-схеми програм. Задача еквівалентності програм належить до числа найбільш важливих задач теорії програмування. Особливе значення задачі еквівалентності обумовлено тим, що саме до цієї проблеми еквівалентних перетворень зводиться більшість задач оптимізації, верифікації та аналізу програм [3]. Тому розробка і впровадження ефективних алгоритмів перевірки еквівалентності програм справляє помітний вплив на підвищення продуктивності та якості багатьох інструментальних засобів аналізу і перетворення програм. В останні роки поряд із задачами підвищення ефективності і надійності програм не меншої актуальності набула задача своєчасного виявлення стороннього коду, яка включає виявлення недекларованих можливостей програмних продуктів, а також виявлення та усунення програм-вірусів.

В обох випадках такі графи можуть містити велику кількість вершин, тому виникає завдання зменшення їх кількості зі збереженням необхідних властивостей графа. Ця задача відома як задача мінімізації.

У даній роботі розглядається проблема мінімізації орієнтованих детермінованих графів з поміченими вершинами. Ця проблема дозволяє знаходити еквівалентні графи, за допомогою яких можна представити різні системи. Аналіз графів проводиться методами, аналогічними методам теорії автоматів. Ці методи створені для графових систем, які не є кінцевими автоматами, але в деякому розумінні автоматоподібними системами.

Проблема мінімізації полягає в знаходженні розбиття всіх вершин графа на класи еквівалентних. В [1] запропоновано алгоритм мінімізації графа часової складності $O(2^n)$ у загальному випадку і $O(n^2)$ для так званих детермінованих [3] графів.

1 Постановка задачі

Нехай G = (V, E, M, μ) – кінцевий, помічений, орієнтований граф, без петель і кратних дуг [1], де V – множина вершин, $E \subseteq V \times V$ – множина дуг, M – множина номерів поміток, μ: S→M – функція розмічування вершин, $E(v) = \{u \in V|(v,u) \in E\}$ – множина наступників вершини v і $E^{-1}(v) = \{u \in V|(u,v) \in E\}$ – множина її попередників. Кінцеву послідовність вершин $p = g_1…g_k$, таку, що g_{i+1} є $E(g_i)$, 1<i< = k, назвемо шляхом у графі G. Слово $\mu(p) = \mu(g_1)… \mu(g_k)$ назвемо відміткою шляху p. Позначку будь-якого шляху, що виходить з вершини g є V, будемо називати словом, породженим вершиною g. Мову Lg визначимо як множину всіх слів, породжених вершиною g. Вершини g, h – назвемо еквівалентними, якщо Lg = Lh, і відмінними в іншому випадку. Граф G називається приведеним, якщо всі його вершини попарно відрізняються. Послідовність вершин $g_i...g_k$ називається ациклічним підграфом графа тоді і тільки тоді, коли для будь-якої g_j вершини послідовності є шлях тільки у вершини $g_{j+1}...g_k$. Вершини ациклічного підграфа природно ранжуються за рівнями: 1) якщо v – лист в G, то рівень ρ(v) дорівнює 0; 2) $\rho(v) = i + 1$, якщо $\rho(u) \le i$ для всіх $u \in E(v)$ и $\rho(u) = i$ для деякої $u \in E(v)$. Число вершин в E(v) назвемо ступінню вершини v та позначимо st(v). Через μ|E(v)| позначимо множину всіх відміток з E(v). Через |V| позначимо потужність множини V. Відношення $\pi_1 \subseteq \pi_2$ має на увазі, що кожний клас із π_1 повністю входить у клас з π_1. Клас, який не може бути розбитий, назвемо остаточно розбитим.

Невизначені поняття загальноприйняті і можуть бути знайдені в [4]. У даній роботі задача мінімізації зводиться до знаходження грубішого розбиття π, задовольняє умові: $\pi \subseteq \pi_0$, де π_0 – так зване початкове розбиття, причому вершини a і b знаходяться в одному його класі, якщо $\mu(a) = \mu(b)$ та $|\mu[E(a)]| = |\mu[E(b)]|$, а також π є розбиттям з властивістю підстановки, якщо a, b в одному класі π, і (a,c), (b,d) ∈E, μ(c) = μ(d), то c,d також входять в один клас π.

2 Метод і алгоритм знаходження еквівалентних вершин

Метод рішення складається в тому, що, виходячи з π_0, будується послідовність розбиття $\pi_1 \supseteq \pi_2, ..., \pi_i, ..., \pi_k \supseteq \pi_{k+1}$. Перехід від π_i до π_{i+1} відбувається наступним чином: спочатку обробляються (мінімізуються) вершини, що належать ациклічним підграфам [5], далі вони оброблятися не будуть. Необроблені вершини мінімізуються методом, що є модифікацією алгоритму Хопкрофта [6]. Модифікація полягає в тому, що на повторну обробку поміщаються номери класів не з меншою кількістю вершин, а з меншою кількістю попередників для класу.

Ідея алгоритму

1 Будуємо початкове розбиття π_0.

2 Мінімізуємо всі ациклічні підграфи [5], при цьому вершини, які належать підграфам, більше не будуть оброблятися (див. нижче пункт 2 алгоритму).

3 Мінімізуємо граф, що залишився алгоритмом [6] (див. нижче пункт 3 алгоритму).

Алгоритм

ВХІД: граф G.

ВИХІД: розбиття на класи еквівалентних вершин p' = (B[1], ...,B[q]) множини V.

1 i = 0. Будуємо початкове розбиття π_i, номера його класів, що містять вершини рівня 0, поміщаємо в множину WAIT0 (активні класи), інші номери класів поміщаємо в множину WAIT. Для всіх v ступінь st(v) = |E(v)|.

2 Поки WAIT0 не пусте, виконуємо наступне [5]:

2.1 Для всіх вершин v, що належать активним класам, зменшуємо ступінь вершин u $\in E^{-1}(v)$, тобто st(u) = st(u) – 1.

2.2.1 Для кожного активного класу K $\in$ WAIT виконуємо наступне: будуємо $K' = \bigcup_{v \in K} E^{-1}(v)$ і перетинаємо K' з класами розбиття π_i, що дає розбиття π_{i+1}. Якщо клас $K'' \subseteq K'$ та номер $K'' \in$ WAIT, то його номер видаляємо з WAIT. При цьому номер K видаляється з WAIT0.

2.2.2 В WAIT0 помістимо номера класів з π_{i+1}, які містять вершині v з $\rho(v) = i+1$, та st(v) = 0.

3. Поки WAIT не пусте, виконуємо наступне:

3.1 Вибрати та видалити найменший клас K із WAIT.

3.2 Будуємо $K' = \bigcup_{v \in K} E^{-1}(v)$ та перетинаємо K' з класами розбиття π_i, що дає розбиття π_{i+1}. Якщо при перетині K' з π_i розбивається який-небудь клас K'' на класи K'' і KN, то в множину WAIT помістимо номер наступного класу: якщо номер $K'' \in$ $\in$ WAIT, то номер нового класу KN, інакше номер класу, що задовольняє умові min(|$E^{-1}(K'')$|,| E^{-1}(KN)|).

3.3 Алгоритм зупиняється, якщо WAIT=$\varnothing$.

Пояснення алгоритму

Алгоритм складається з двох етапів. На початку будується розширене початкове розбиття π_0. Номери класів, що містять вершини зі ступенем 0, помістимо в множину WAIT0, інші – в множину WAIT.

На першому етапі мінімізуються всі ациклічні підграфи. У множину WAIT0 можуть потрапити тільки остаточно розбиті класи, спочатку це класи, які містять листя графа. Так як у листя немає наступників, то їх мова складається з однієї літери, і ці класи ми отримуємо після початкового розбиття. Далі для всіх вершин v $\in$B[i], i $\in$ WAIT0 обробляємо попередників $g \in E^{-1}(v)$ і st(g) = st(g) – 1, зменшуємо ступінь на 1, таким чином ми видаляємо нижній рівень ациклічних підграфів. Цикл пункту 2 виконується, поки будуть з'являтися нові вершини зі ступенем, рівним 0. Якщо таких вершин немає, тоді всі ациклічні підграфи були мінімізовані, і алгоритм переходить на другий етап, в якому не оброблені класи мінімізуються загальним алгоритмом. Всі номери класів, які можуть бути розбиті або можуть розбити будь-який клас, перебувають у множині WAIT. Для кожного класу з множини WAIT видаляємо номер класу з множини, будуємо множину попередників E^{-1}(B) для обраного класу B. Перетинаємо попередників з існуючими класами. Якщо клас K не повністю перетинається E^{-1}(B), значить клас розбивається на K і KN. Якщо номер K вже міститься в WAIT, то в множину WAIT помістимо номер класу KN, інакше в множину поміщається номер того класу, у якого попередників менше. Алгоритм закінчує свою роботу, коли в множині WAIT не буде жодного класу.

3 Обґрунтування і складність алгоритму

Теорема. Алгоритм коректний, тобто завершується за кінцеве число кроків і будує шукане розбиття π за $O(m \cdot n \cdot \log(n))$ кроків.

Доказ. Покажемо, що алгоритм через кінцеве число кроків зупиниться. Початкове розбиття вершин виконується за n кроків, n – кількість вершин графа. У пункті 2 алгоритму вершини обробляються тільки один раз, отже, він закінчить свою роботу за

кінцеве число кроків. У пункті 3 номер класу може потрапити в множину WAIT тільки при розбитті якогось класу. Так як клас еквівалентності не може містити менше 1 вершини, тоді розбиттів у графі буде не більше n. Всі етапи алгоритму виконуються за кінцеве число кроків, отже, через кінцеве число кроків алгоритм зупиниться.

Покажемо, що алгоритм працює коректно, тобто π є грубе розбиття вершин на класи еквівалентних, π має властивість підстановки. Включення $\pi \subseteq \pi_0$ виконується з побудови, так як при роботі алгоритму класи розбиваються, а не об’єднуються. π є розбиттям з властивістю підстановки, так як якщо вершини v і u належать одному класу еквівалентності, то $K' = E^{-1}(u) \bigcup E^{-1}(v)$ не розбиває класи розбиття π, інакше вони були б розбиті раніше. Покажемо, що розбиття π найгрубіше. Нехай існує більш грубе розбиття π'. $\pi' \not\subseteq \pi_0$ з властивістю підстановки, кількість класів в π' буде менше ніж в π, значить існують вершини v і u, які в π належать різним класам еквівалентності, а в π' – одному. Але це неможливо, так як u та v при побудові π знаходяться у різних класах, тобто не еквівалентні, та π' не може включатися в π. Таким чином, алгоритм будує шукане розбиття π.

Оцінимо часову складність алгоритму в найгіршому випадку. Пункт 1 алгоритму виконується за N кроків [5]. Нехай кількість вершин, які належать ациклічним підграфам, дорівнює k. Покажемо, що пункт 2 виконується за $\sum|E^{-1}(v)|$ кроків, де v – вершини, які належать ациклічним підграфам, що не перевищує O(|E|). Розглянемо складність виконання всього пункту 2 алгоритму, в якому обробку проходять тільки остаточно розбиті вершини. Це означає, що кожна вершина обробляється в пункті 2 тільки 1 раз. Для кожної вершини v оброблюваного рівня ми зменшуємо ступінь вершин $u \in E^{-1}(v)$, це вимагає числа кроків, рівного кількості дуг, що входять до v. Під обробкою рівня ациклічних підграфів розуміється зменшення ступеня попередників вершин цього рівня і побудова розбиття π_i. Після обробки цей рівень графа видаляється, значить, видаляються розглянуті дуги, тому кожна дуга обробляється в циклі лише 1 раз. У циклі пункту 2.2.1 дуги розглядаються 1 раз, і виконується константне число мінімальних операцій додавання, видалення чисел у множині, виділення пам’яті для тимчасових класів, які містять вершини. Цих класів при обробці одного рівня графа не може бути більше числа переглянутих дуг, так як клас повинен містити хоча б 1 вершину. Сумарне число вершин, які можна додавати в множини, так само не перевищує числа дуг, так як одна вершина не може одночасно знаходитися в двох класах еквівалентності. У пункті 2.2.2 виконуються операції додавання і видалення номерів класів у множину WAIT0. У цій множині можуть бути тільки номери остаточно розбитих класів, це означає, що за час роботи алгоритму буде не більше k доповнень і вилучень.

На другому етапі обробляються вершини, які не належать остаточно розбитим класам. Таких вершин буде n–k. На другому етапі виконується 3 пункт алгоритму. Основний цикл виконується |WAIT| разів. Вважаємо, що спочатку в |WAIT| = p.

Клас може потрапити в цю множину тільки при розбитті. Кількість розбиттів може бути не більше n–k–p, так як кожен клас не може містити менше 1 вершини. Отже, цикл пункту 3 буде виконуватися (n–k–p)+p = n–k раз. Для всіх вершин v_j, у яких $\mu(v_j) = a$ $\sum|E^{-1}(v_j)| \le n$. Так як класи не можуть містити однакових вершин, тоді $\sum|E^{-1}(B[i])| \le n$, i – номери класів з відміткою a і належать множині WAIT. Так як різних відміток у графі – m, тоді кількість кроків, необхідних для обробки p класів, не перевищує mn кроків. Якщо розбився клас K, номер якого вже міститься в WAIT, значить складність алгоритму не збільшується, так як $K = K1 \cup K2$, а отже, за властивістю детермінованих графів $E^{-1}(K) = E^{-1}(K1) \cup E^{-1}(K2)$. Якщо розбивається клас вже оброблений та його номер не міститься в WAIT, тоді в множину поміщається клас, у якого менше попередників. Отже, клас потрапляє на обробку тільки тоді, коли попередників у ньому буде як мінімум в 2 рази менше і так далі на зменшення. Таким чином складність другого етапу

дорівнює mnlog(n). Загальна складність алгоритму дорівнює O(n) = (k+nmlog(n)) та рівняється O (nmlog(n)). Якщо ми будемо мінімізувати дерево, то другий етап роботи алгоритму ніколи не буде виконуватися, і тоді складність всього алгоритму буде дорівнювати O(n). А для ациклічних графів – O(mn).

Теорема доведена.

Приклад

Розглянемо виконання даного алгоритму на графі (рис. 1). Початкове розбиття для графа дорівнює π_0 = {1{1,2,9,12,14}, 2{7,16,10}*, 3{3,4,5,13} 4{6,8,11,15}}. Класи з * далі розбиватися не можуть. Зараз – це клас 2, в якому всі вершини є листям. В WAIT0 помістимо номера класів, які містять листя, WAIT0 = {2}. Обробляємо активні класи та отримуємо наступний рівень графа – це вершини {6,9,15} та нове розбиття π_1 = {1{1,2,12,14}, 2{7,16,10}*, 3{3,4,5,13}, 4{8,11}, 5{6,15}*,6{9}*} на класи еквівалентних вершин. Активними стають вершини першого рівня WAIT0 = {5{6,15}, 6{9}}. Оброблюємо активні класи та отримуємо новий рівень графа, це вершина {8} та нове розбиття π_2 = {1{1,2,12,14}, 2{7,16,10}*, 3{3,4}, 4{11}, 5{6,15}*, 6{9}*, 7{8}*, 8{5,13}}. Отримуємо вершини другого рівня WAIT0 = {7{8}}. Обробляємо активні класи, отримаємо розбиття π_3 = {1{1,2,12,14}, 2{7,16,10}*, 3{4}, 4{11}, 5{6,15}*, 6{9}*, 7{8}*, 8{5,13}, 9{3}}. Вершин нового рівня не отримуємо, значить, оброблені всі ациклічні підграфи. Для пошуку вершин за ступінню 0 потрібно 16 кроків. А для обробки ациклічних підграфів потрібна кількість кроків, рівна кількості дуг, які входять в ациклічні вершини, що дорівнює 7.

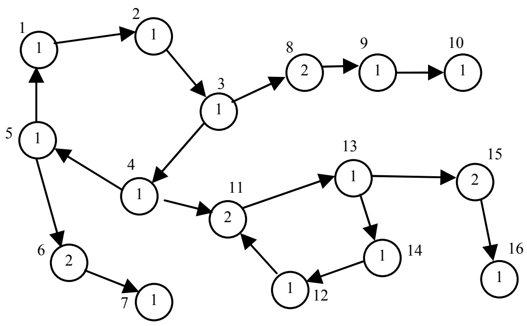


Рисунок 1 – Загальний вид графа

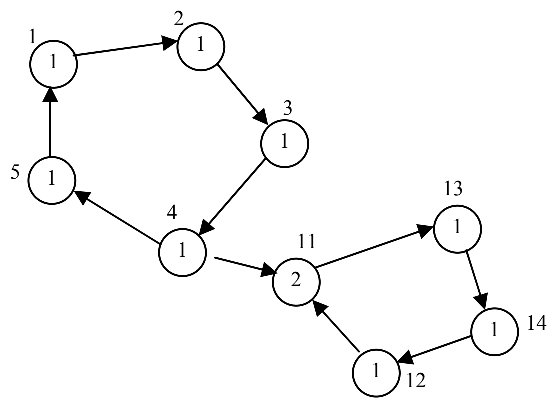


Рисунок 2 – Граф без ациклічних підграфів

На обробку цього етапу потрапляє граф без ациклічних підграфів (рис. 2) та у множину WAIT помістимо не остаточно розбиті класи, WAIT=. {1{1,2,12,14}, 8{5,13}, 3{4}, 9{3}, 4{11}}. Оброблюємо класи з множини WAIT послідовно, завжди вибираємо останній клас. В результаті отримуємо розбиття π = {1{1}, 2{7,16,10}*, 3{4}, 4{11}, 5{6,15}*, 6{9}*, 7{8}*, 8{5}, 9{3}, 10{12}, 11{14}, 12{13}, 13{2}}. На виконання другого етапу потрібно 10 кроків. В сумі алгоритму потрібно T(n) = 16+7+10 = 33. Теоретична оцінка складності алгоритму дорівнює O(m*n*log n) = 16*2*4 = 128.

Запропонований алгоритм для деяких видів графів дає лінійну часову складність. Це можливо у випадках, коли обробляється ациклічний граф. У цьому випадку буде виконуватися тільки перший етап алгоритму. І тимчасова складність буде залежати від виду ациклічних графів. Розглянемо тимчасову складність для деяких видів ациклічних графів. Для дерев – O(n), для детермінованих ациклічних графів – O(n*m).

Висновки

Запропоновано новий метод мінімізації орієнтованих детермінованих графів із поміченими вершинами. Часова асимптотична складність мінімізації детермінованих графів загального виду – O(m*n*logn), детермінованих ациклічних графів – O(m*n), дерев – O(n).

Подяки

Автор вдячний своєму науковому керівнику І.С. Грунському за постановку задачі і допомогу в роботі.

Література

1. Сапунов С.В. Анализ графов с помеченными вершинами: дисс. … канд. физ.-мат. наук : 27.09.07. / Сапунов С.В. – Донецк, 2007. – 150 с.
2. Map validation and Robot Self-Location in a Graph-Like World / J. Dudek, M. Jenkin, E. Milios, D. Wilkes // Robotics and Autonomous Systems. – 1997. – Vol. 22(2). – P. 159-178.
3. Касьянов В.Н. Графы в программировании: обработка, визуализация и применение / В.Н. Касьянов, В.А. Евстигнеев. – БХВ – Петербург, 2003.
4. Чепурко В.А. Минимизация ориентированных ациклических графов с отмеченными вершинами: материалы VI международной научно-практической конференции / В.А. Чепурко // (Днепропетровск, 12 – 14 ноября 2008 г.). – Днепропетровск : Днепропетровский национальный университет им. Олеся Гончара, 2008. – С. 331-332.
5. Ахо А. Построение и анализ вычислительных алгоритмов / Ахо А., Хопкрофт Дж., Ульман Дж. – Мир, 1979. – 536 с.
6. Чепурко В.А. Минимизация ориентированных графов с отмеченными вершинами / В.А. Чепурко // «Информатика и компьютерные технологии» : материалы V юбилейной Международной научно-технической конференции студентов, аспирантов и молодых учёных (Донецк 24 – 26 ноября 2009 г.). – Донецк : Донецкий национальный технический университет, 2009. – С. 502-509.

В.А.Чепурко

Минимизация ориентированных детерминированных графов с ациклическими подграфами

Графы с помеченными вершинами являются одной из основных моделей в рассмотрении проблем, связанных с анализом оперативной среды и агентами, движущимися по ней, а также проблем, связанных с проверкой программ. Задача минимизации заключается в нахождении разбиения всех вершин графа на классы эквивалентных вершин. Предложен новый алгоритм минимизации для графов с отмеченными вершинами. Алгоритм выполняет правильное разбиение на классы эквивалентных вершин.

V.A. Chepurko

Reduction of Graphs with Marked Vertices and Acyclic Components

Graphs with labeled vertices are one of the main models in consideration of problems associated with the analysis of the operating environment with agents moving on it as well as problems associated with the validation of program. In both cases, these graphs may contain a large number of vertices, so the problem of reducing their number with retain all properties of the graph is arisen. The reduction problem is to find a partition of all vertices of the graph into classes of equivalent states. A new algorithm for graphs reducing is proposed. It consist of next steps. The algorithm builds a correct partition into classes of equivalent states.

Стаття надійшла до редакції 29.06.2010.